1. INTRODUCCIÓN

El DML (Lenguaje de Manipulación de Datos) es una de las partes fundamentales del lenguaje SQL. Lo forman las instrucciones capaces de modificar (añadir, cambiar o eliminar) los datos de las tablas.

> Al conjunto de instrucciones DML que se ejecutan consecutivamente, se le llama transacción. Lo interesante de las transacciones es que podemos anularlas, ya que forman una unidad lógica de trabajo que hasta que no se acepten, sus resultados no serán definitivos.

En todas las instrucciones DML, el único dato devuelto por el sistema es el número de filas que se han modificado al ejecutar la instrucción.

Antes de introducirnos en el estudio de las instrucciones INSERT, UPDATE y DELETE, es necesario conocer cómo el SGBD gestiona las instrucciones de inserción, eliminación y modificación que podamos ejecutar, ya que existen dos posibilidades de funcionamiento:

- Que queden automáticamente validadas y no haya posibilidad de echar atrás. En este caso, los efectos de toda instrucción de actualización de datos que tenga éxito son automáticamente accesibles desde el resto de conexiones de la base de datos.
  (La opción que tenemos en las máquinas virtuales por defecto)

- Que quedan en una cola de instrucciones, que permite echar atrás. En este caso, se llama que las instrucciones de la cola están pendientes de validación, y el usuario debe ejecutar, cuando lo cree conveniente, una instrucción para validarlas (llamada COMMIT) o una instrucción para echar atrás (llamada ROLLBACK).

Este funcionamiento implica que los efectos de las instrucciones pendientes de validación no se ven por el resto de conexiones de la base de datos, pero sí que son accesibles desde la conexión donde se han efectuado. Al ejecutar la COMMIT, todas las conexiones acceden a efectos de las instrucciones validadas. En caso de ejecutar ROLLBACK, las instrucciones desaparecen de la cola y ninguna conexión (ni la propia ni el resto) accede a los efectos correspondientes, es decir, es como si nunca hubieran existido.

Estos posibles funcionamientos forman parte de la gestión de transacciones que proporciona el SGBD y que es necesario estudiar con mayor detenimiento. Pero a la vez de ejecutar

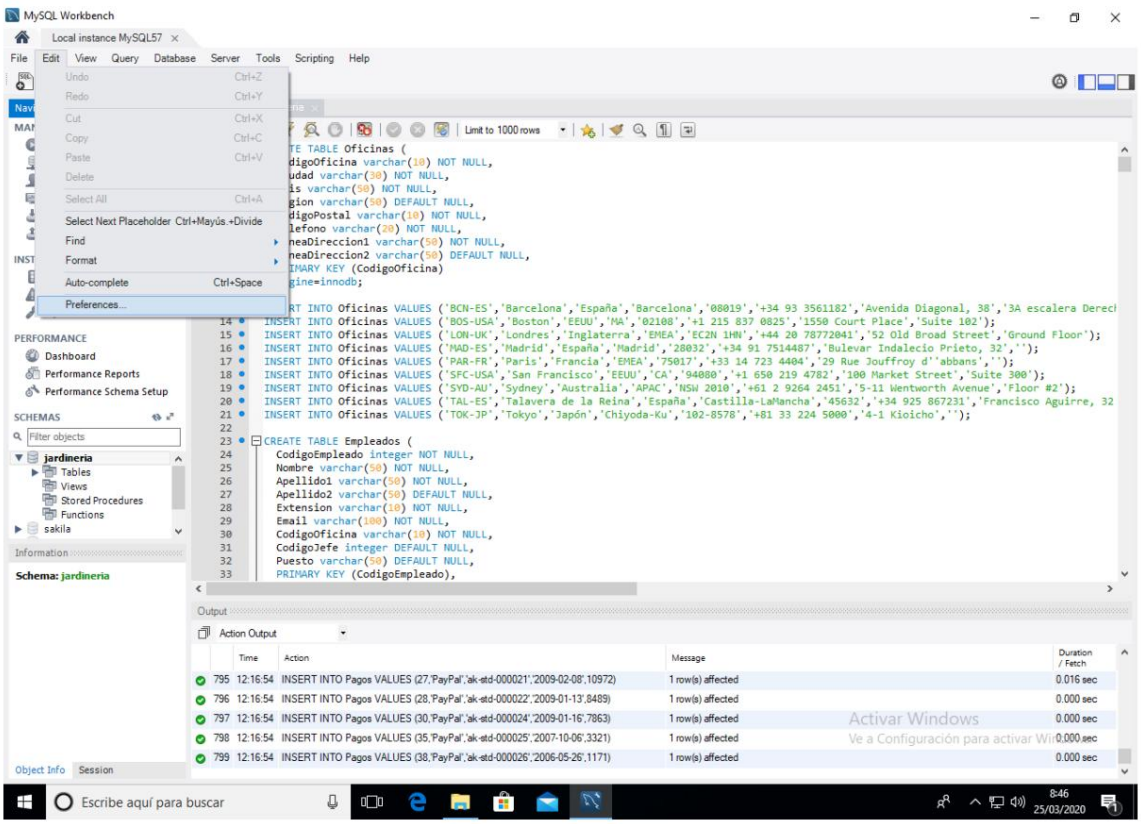
instrucciones INSERT, UPDATE y DELETE debemos conocer el funcionamiento del SGBD para poder actuar en consecuencia.

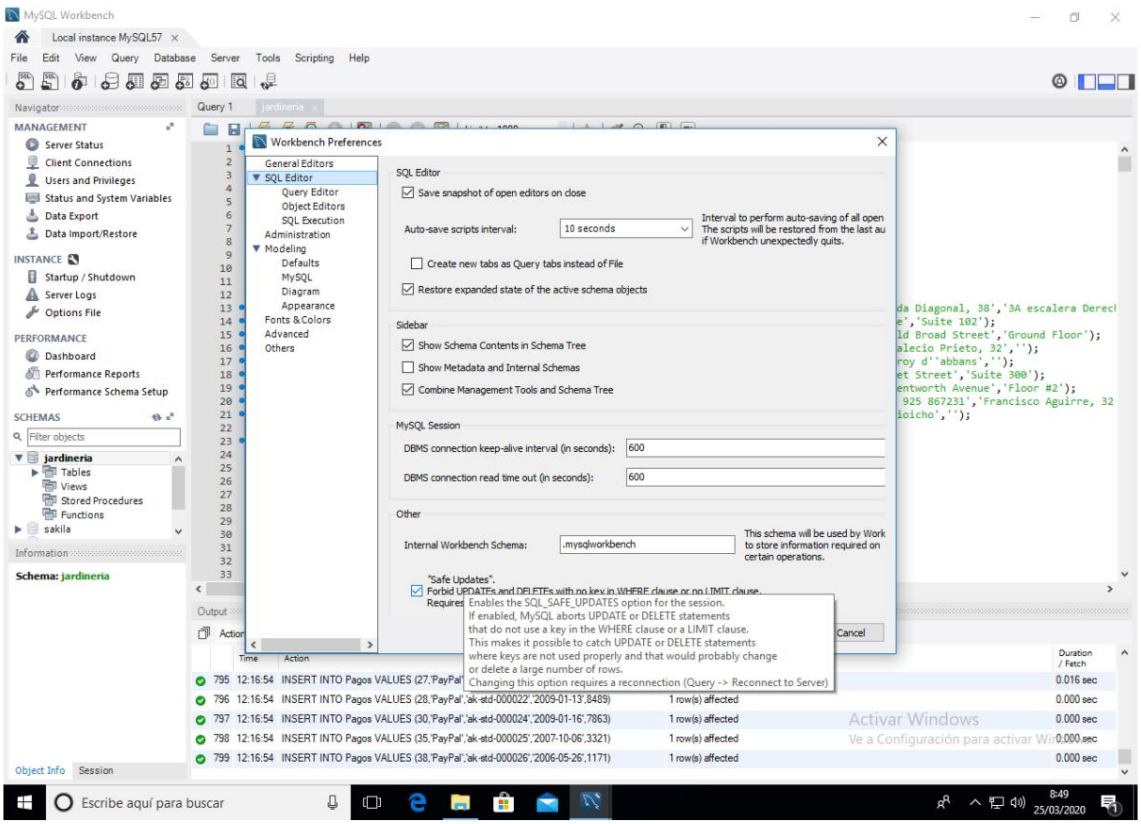
Así, por ejemplo, un SGBD MySQL funciona con validación automática después de cada instrucción de actualización de datos no se indica lo contrario y, en cambio, un SGBD Oracle funciona con la cola de instrucciones pendientes de confirmación.

En cambio, en MySQL, si se desea desactivar la opción de autocommit que hay por defecto, será necesario ejecutar la siguiente instrucción:

```
ESTABLECER confirmación automática = 0;
```

En nuestro caso, mantendremos la opción por defecto de MySQL y desactivaremos la opción de MySQL "Safe Updates". Esto permitirá que nuestras sentencias UPDATE o DELETE se ejecutan sin seguridad (aunque sea eliminar todos los registros de una tabla) y podamos realizar las prácticas de manera fluida.





Hay que quitar la opción por defecto que es que 'Safe updates' esté seleccionado. Esta opción no afecta a INSERT, pero sí a UPDATE y DELETE.

Recuerda reconectar con el servidor para que la nueva configuración surta efecto. Dos opciones:

- Cierra la conexión y vuelve a abrirla
- Reconecta a través de la opción en el menú Query/Reconnect to Server

2. LENGUAJE DE MANIPULACIÓN DE DATOS: DML

Una vez que se han creado de manera conveniente las tablas, el siguiente paso consiste en insertar datos en ellas, es decir, añadir tuplas. Durante la vida de la base de datos será necesario, además, borrar determinadas tuplas (files) o modificar los valores que contienen. Los comandos de SQL que se estudiarán en este apartado son INSERT,
UPDATE y DELETE. Estos comandos pertenecen al DML.

3. INSERCIÓN DE DATOS

El comando INSERT de SQL permite introducir datos en una tabla o en una vista de la base de datos (las vistas no las consideraremos). La sintaxis de INSERT es la siguiente:

```
INSERTAR EN nombre_taula [(columna1 [, columna2]...)]
VALUAS ({expresión|valor} [, {expresión|valor}] ... )
                [, ({expresión|valor} [, {expresión|valor}] ... )]...;
```

En ella podemos distinguir las palabras reservadas INSERT INTO seguidas del nombre de la tabla en la que guardaremos los nuevos datos.

Opcionalmente podemos poner entre paréntesis los nombres de los campos en en los campos en los que queremos poner valores, si no los incluimos, se tendrán que colocar todos los valores en el mismo orden en que fueron creados los campos (es el orden de columnas según las devuelve el comando DESCRIBE), porque de lo contrario se producirá un error si los tipos no coinciden o se almacenará la información de forma errónea en caso de que los tipos de datos de los campos sean compatibles.

Los datos de las filas a introducir, se especifican después del apartado VALUES.

Cada nueva fila requiere un paréntesis con sus valores. Se pueden introducir varias filas con un solo INSERT separando cada paréntesis con comas.

En cada nueva fila los valores deben corresponderse con el orden de las columnas.

Supongamos que tenemos el siguiente diseño físico de una tabla (Compruébalo en el SGBD) :

```
CREAR DIAGRAMA DE PRUEBA ;
PRUEBA DE USO ;
CREAR PLANTILLAS DE TABLA (
  COD entero CLAVE PRIMARIA,
  NOM VARCHAR(50) NO NULO,
  LOCALIDAD VARCHAR(50) PREDETERMINADO 'Écija',
  fecha DATANAIX );
```

Como se puede observar, los campos LOCALIDAD y DATANAIX no son obligatorios por lo que podríamos optar por no cumplimentarlos. En ese caso, deberíamos definir en INSERT cuáles campos rellenaremos y en qué orden. Por ejemplo:

```
INSERT INTO EMPLEADOS(NOMBRE, COD) VALUES ('Ana', 1);
```

```
/* También podemos introducir varias filas */

INSERTAR EN EMPLEATS(NOMBRE, COD) VALORES ('Pep', 2), ('Gemma', 3);

SELECCIONAR * DE EMPLEATS;
```

Si un campo tiene un valor predeterminado y INSERT no especifica un valor para el campo,
Se les asigna el valor predeterminado.

También podemos no especificar los campos a llenar y en este caso debemos asignar valores a todos los campos en el orden de creación. Por ejemplo:

```
INSERTAR EN LOS VALORES EMPLEATS (4, 'Joan', 'Osuna', '1999-01-30'),
(5, 'Juan', PREDETERMINADO, NULO), (3, 'Sara', NULO, NULO);
```

Como vemos, se introducen datos en todos los campos en el orden de creación.

También, sería equivalente, podríamos hacerlo con un solo INSERT:

```
INSERTAR EN EMPLEATS VALORES (1, 'Pepe', 'Osuna', '1999-01-30'), (2,
'Juan', POR DEFECTO, NULO), (3, 'Sara', NULO, NULO);
```

Es obligatorio introducir valores para los campos COD y NOMBRE. Estos campos no pueden tener valor NULL.

Columnas auto_increment

Las columnas auto_increment tienen por objeto establecer una clave cuando no tenemos en una mesa una clave clara.

Normalmente se describen las describimos así:

Clave principal de incremento automático de enteros de NomCamp

Aunque también podrían tener la restricción UNIQUE en lugar de la de Primary Key.

Su objeto es que el sistema asigne un valor al campo de modo que siempre asigna un valor diferente (según una secuencia).

Se puede asignar un valor a una columna auto_incremento aunque desvirtuaría el su sentido por lo que normalmente, cuando se hace una inserción, o no se asigna valor al campo o

se le asigna el valor NULL. En ambos casos, el sistema asignará al campo el valor que corresponda. Ejemplo:

```
crear tabla prova3 (
num entero auto_increment clave primaria,
nombre varchar(10));
insertar en prova3 valores(null,'Pere');
insertar en prova3 (nom) valores('Anna');
seleccione * de prova3;
```

Inserción de datos obtenidos de una consulta

También es posible insertar datos en una tabla que hayan sido obtenidos de una consulta realizada en otra mesa u otras tablas. Su forma es:

```
INSERTAR EN tabla [(columna1 [, columna2]...)]
SELECCIONAR ...
```

Debe respetarse lo dicho anteriormente respecto a los campos. La consulta SELECT debe devolver la misma cantidad y tipos de campos compatibles con los definidos en la mesa.

Como en el caso anterior, podemos optar por especificar o no los campos a llenar. Por ejemplo, suponiendo que disponemos de una tabla BUS_OSUNA con el siguiente diseño:

```
CREAR TABLA BUS_OSUNA (
  NUM entero CLAVE PRIMARIA,
  NOMBRE VARCHAR(50));
```

```
INSERTAR EN BUS_OSUNA
SELECT COD, NOM FROM EMPLEATS
DONDE UBICACION='Osuna';

SELECCIONAR * DE BUS_OSUNA;
```

También podemos indicar los campos a insertar, teniendo en cuenta que, en este caso los campos de la que no aceptan valores NULL. Por lo tanto, es obligatorio introducir valores para a ellos:

```
INSERTAR EN BUS_OSUNA (NUM)
SELECCIONE COD DE EMPLEADOS
WHERE LOCALITAT IS NULL;
SELECCIONAR * DE BUS_OSUNA;
```

INSERTAR RESUMEN

Podemos insertar en una mesa el resultado de una consulta sobre otra mesa. En este caso normalmente se insertarán varias filas con una sola sentencia. Utilizaremos el siguiente formato:

```
INSERTAR EN NomTaula [(NomColumna [,NomColumna...] ) ]
SELECCIONAR FormatoSeleccionar
```

Notación: la lista de columnas en las que insertamos va es opcional, por lo que va entre corchetes.

En el formato anterior podemos destacar:

· La lista de columnas es opcional pero deberá especificarse cuando las columnas que devuelve la consulta no coinciden en número o en orden con las columnas de la tabla destine.

· La consulta puede ser cualquier comando de selección válido siempre que exista una correspondencia entre las columnas devueltas y las columnas de la tabla destine, o la lista de columnas.

EJERCICIOS BD JARDINERÍA

Cree las tablas PedidosHistorico y DetallePedidos_historico con los campos y las restricciones siguientes que se corresponden con los de Pedidos y DetallePedidos.

- Para PedidosHistorico publicaremos un campamento, FechaIntro, no ingresaremos la fecha actual.
- En el caso de DetallePedidos_historico, la clave ajena será hacia Historial de pedidos.

```
CREATE TABLE PedidosHistorico (
  CodigoPedido entero NO NULO,
  FechaPedido fecha NO NULA,
```

```
  FechaEsperada fecha NO NULA,
  FechaEntrega fecha DEFAULT NULL,
  CodigoCliente entero NO NULO,
  Fecha de introducción de datos,
  PRIMARY KEY (CodigoPedido),
  RESTRICCIÓN pedHist_Client CLAVE EXTERNA (CodigoCliente)
REFERENCES Clientes (CodigoCliente)
) motor=innodb;
```

```
CREATE TABLE DetallePedidosH (
  CodigoPedido entero NO NULO,
  CodigoProducto varchar(15) NOT NULL,
  Cantidad integer NOT NULL,
  PrecioUnidad numérica(15,2) NOT NULL,
  NumeroLinea smallint NO NULO,
  PRIMARY KEY (CodigoPedido,CodigoProducto),
  RESTRICCIÓN HDetallePedidos_PedidoFK CLAVE EXTERNA
(CodigoPedido) REFERENCES PedidosHistorico (CodigoPedido),
  RESTRICCIÓN HDetallePedidos_ProductoFK CLAVE EXTERNA
(CodigoProducto) REFERENCES Productos (CodigoProducto)
)motor=innodb;
```

(Los ejercicios están en castellano porque la BD está en castellano)

1) Introducid en la tabla Empleados 3 empleados nuevos. Recordad que tendréis que asignarlos a una OFICINA válida.

2) Introducid en la tabla CLIENTES tres clientes que estén asignados a cada uno de los nuevos empleados creados en el ejercicio anterior.

3) Introducid en la tabla Pedidos_historico los pedidos que ya se han servido (tienen fecha de entrega). En el campo DataIntro debe introducirse la fecha actual (la del sistema).

4) Introducid en la tabla DetallePedidos_historico las filas de DetallePedidos

relacionadas con las filas de Pedidos_historico.